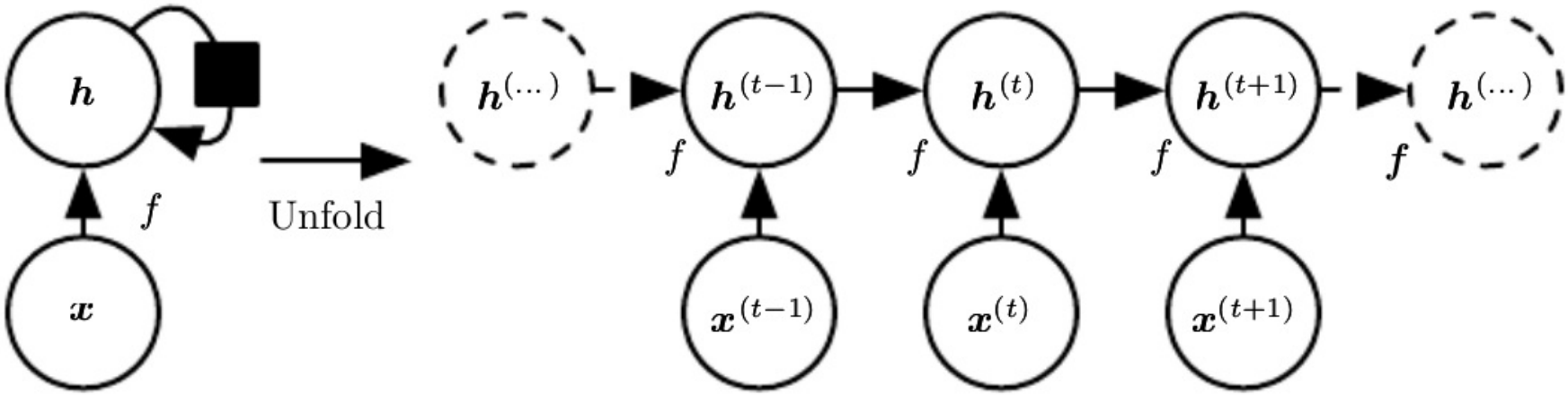
Language Modeling and Recurrent Neural Networks (RNNs)

Language modeling and RNNs

We will start with slides 24-72 on the same topic from the 2021 Stanford CS224N class "Deep Learning for Natural Language Processing":

- `http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture05-rnnlm.pdf`

Notation



RNNs can be represented in two ways. First, as in the left of the above figure, where the black square indicates a delay of a single time step. Secondly, as in the right of the above figure, where the same network is seen as an **unfolded computational graph**, where each node is now associated with one particular time instance.

Note

In slide 53 of the Stanford slides, we have

$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \mathbf{W}_h}\Big|_{(i)}.$$

In the above expression:

$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h}\Big|_{(i)} = \frac{\partial J^{(t)}}{\partial \hat{\mathbf{y}}^{(i)}} \frac{\partial \hat{\mathbf{y}}^{(i)}}{\partial h^{(i)}} \frac{\partial h^{(i)}}{\partial W_h}$$

Notice also that $h^{(i)}$ depends on $h^{(i-1)}$ so we need to **backpropagate through time** to compute the gradient.

See also the following article for interesting discussion and examples: `https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f`.

Training RNNs - Forward pass

The **forward pass** of an RNN is the same as that of a feed-forward network with a single hidden layer, except that activations arrive at the hidden layer from both the current external input and the hidden layer activations from the previous timestep.

Consider a length T input sequence $\mathbf{x}$ presented to an RNN with I input units, H hidden units, and K output units. Let $\mathbf{x}_i^t$ be the value of input i at time t, and let in_j^t and a_j^t be respectively the network input to unit j at time t and the activation of unit j at time t. (Compared with the notation of the Stanford slides, we do not use brackets around the superscript t.)

For the hidden units, we have

$$in_h^t = \sum_{i=1}^{I} w_{ih}\mathbf{x}_i^t + \sum_{h'=1}^{H} w_{h'h}a_{h'}^{t-1}.$$

Training RNNs - Forward pass (cont'd)

Nonlinear, differentiable activation functions are then applied exactly as for a feed-forward network:

$$a_h^t = g_h(in_h^t)$$

The complete sequence of hidden activations can be calculated by starting at $t = 1$ and recursively applying the above formulas, incrementing t at each step.

Note that this requires initial values a_i^0 to be chosen for the hidden units, corresponding to the network's state before it receives any information from the data sequence. The initial values can be set to zero. However, researchers have found that RNN stability and performance can be improved by using nonzero initial values.

Training RNNs - Forward pass (cont'd)

The network inputs to the output units can be calculated at the same time as the hidden activations:

$$in_k^t = \sum_{h=1}^{H} w_{hk} a_h^t, \text{ for all } k \in K$$

The output activation functions can be sigmoid or softmax as we presented in the Stanford slides.

Training RNNs - Backward pass

Given the partial derivatives of some differentiable loss function L with respect to the network outputs, the next step is to determine the derivatives with respect to the weights.

Two well-known algorithms have been devised to efficiently calculate weight derivatives for RNNs: **real time recurrent learning** (**RTRL**; Robinson and Fallside, 1987) and **backpropagation through time** (**BPTT**; Williams and Zipser, 1995; Werbos, 1990).

We focus on BPTT since it is both conceptually simpler and more efficient in computation time (though not in memory).

Training RNNs - Backward pass (cont'd)

Like standard backpropagation, BPTT consists of a repeated application of the chain rule. The subtlety is that, **for recurrent networks, the loss function depends on the activation of the hidden layer not only through its influence on the output layer, but also through its influence on the hidden layer at the next time step.**

Therefore

$$\Delta_h^t = g'(in_h^t)(\sum_{k=1}^{K} \Delta_k^t w_{hk} + \sum_{h'=1}^{H} \Delta_{h'}^{t+1} w_{hh'})$$

where Δ_j^t is defined to be $\frac{\partial L}{\partial in_j^t}$.

Proof?

Training RNNs - Backward pass (cont'd)

The complete sequence of Δ terms can be calculated by starting at $t = T$ and recursively applying the formula on the previous slide, decrementing t at each step.

Note that $\Delta_j^{T+1} = 0$ for all j, since no error is received from beyond the end of the sequence.

Finally, bearing in mind that the same weights are reused at every time step, we sum over the whole sequence to get the derivatives with respect to the network weights:

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial L}{\partial in_j^t} \frac{\partial in_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \Delta_j^t a_i^t$$

Generating text using RNNs

To be able to generate text using an RNN, we need a strategy for sampling from the probability distribution of words produced by the RNN at each time step (see slide 48 in the Stanford slides).

The recent paper (Holtzman et al., 2020) studies this problem.

Readings

- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes and Yejin Choi. *The Curious Case of Neural Text Degeneration*. Available from `https://arxiv.org/pdf/1904.09751.pdf`.
- Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks.* Springer. 2012. Chapter 3.2. Available from `https://www.cs.toronto.edu/~graves/preprint.pdf`.
- Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning.* MIT Press. 2016. Chapter 10.
- Aurelién Geron. *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow.* 2nd edition. 2019. O'Reilly. Chapter 15.

Note: The present slides are based on the above sources. I took the liberty to take text from these sources verbatim without using quotes.